

CodePro AnalytiX™

for Eclipse®, Rational® and Websphere

tools that make every developer a quality expert



Google: www.google.com

Copyright 2006-2010, Google, Inc.. All rights are reserved. Google is a registered trademark of Google, Inc. and CodePro AnalytiX is a trademark of Google, Inc. All other trademarks are the property of their respective owners.

Introduction

This Evaluation Guide provides an overview of CodePro AnalytiX—an industry-leading software solution that is used to detect and correct the code quality issues typically faced in software development. The CodePro AnalytiX solution is specifically designed for organizations that are striving to solve code quality, code review and code dependency issues. CodePro AnalytiX contains a comprehensive set of software analysis tools composed of a collection of native Eclipse plugins. CodePro AnalytiX seamlessly integrates into any Eclipse-based Java development environment, adding code audit, metrics, test generation, code coverage, and team collaboration features and functions.

Use this guide as a tool to assist you in evaluating CodePro AnalytiX and to illustrate how to use CodePro AnalytiX to solve issues you face every day. The exercises illustrate how to use CodePro AnalytiX to provide solutions in these areas: code audit (manual and dynamic), metrics, JUnit test case generation, and analyzing code dependencies. To make the evaluation more realistic, a project file is provided that should be downloaded and used during the evaluation.

The solutions highlighted in this guide represent only a fraction of the complete CodePro AnalytiX feature set. See the documentation pages included with the product for a listing of all the product features.

Table of Contents

Introduction.....	2
Getting Started	5
How to Use this Guide.....	6
Code Audit.....	7
Manual Audit.....	7
Goals.....	7
Steps	7
Benefits.....	8
Using Different Rule Sets	8
Goals.....	8
Features.....	8
Steps	8
Benefits.....	9
Choosing and Importing Rule Sets.....	9
Goals.....	9
Features.....	9
Steps	9

Benefits.....	9
Dynamic Audit.....	10
Goals.....	10
Features.....	10
Steps.....	10
Benefits.....	11
Custom Rule Set Generation.....	11
Goals.....	11
Features.....	11
Steps.....	11
Benefits.....	15
Using Audit Reports.....	15
Goals.....	15
Features.....	15
Steps.....	15
Benefits.....	18
Similar Code Analysis.....	19
Identifying Similar Code.....	19
Goals.....	19
Features.....	19
Steps.....	19
Benefits.....	22
Metrics.....	23
Computing Metrics.....	23
Goals.....	23
Features.....	23
Steps.....	23
Benefits.....	24
Choosing and Creating Metric Sets.....	25
Goals.....	25
Features.....	25
Steps.....	25
Benefits.....	26
Metric Reports.....	26
Goals.....	26
Features.....	27
Steps.....	27
Benefits.....	28
JUnit Test Generation.....	29
Generating Test Cases.....	29
Goals.....	29
Features.....	29
Steps.....	29
Benefits.....	30
Generating Factory Classes.....	31
Goals.....	31

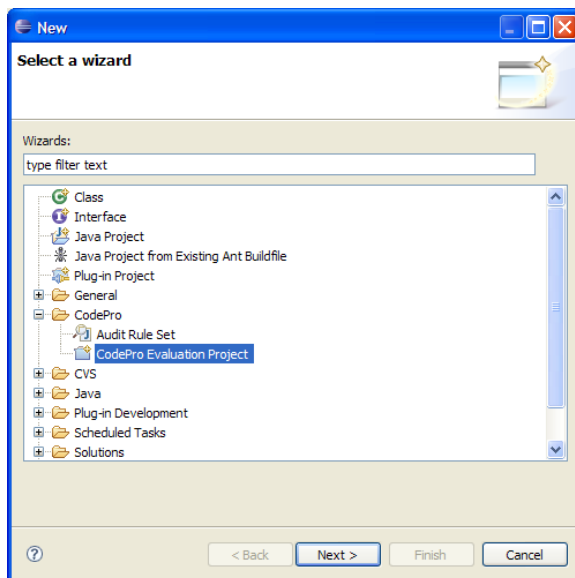
Features.....	31
Steps	31
Benefits.....	32
Code Coverage Analysis	33
Verifying Code Coverage.....	33
Goals.....	33
Features.....	33
Steps	33
Benefits.....	35
Reporting	35
Goals.....	35
Features.....	35
Steps	36
Benefits.....	36
Dependency Analyzer	37
Analyzing Dependencies	37
Goals.....	37
Features.....	37
Steps	37
Benefits.....	39
Summary	40

Getting Started

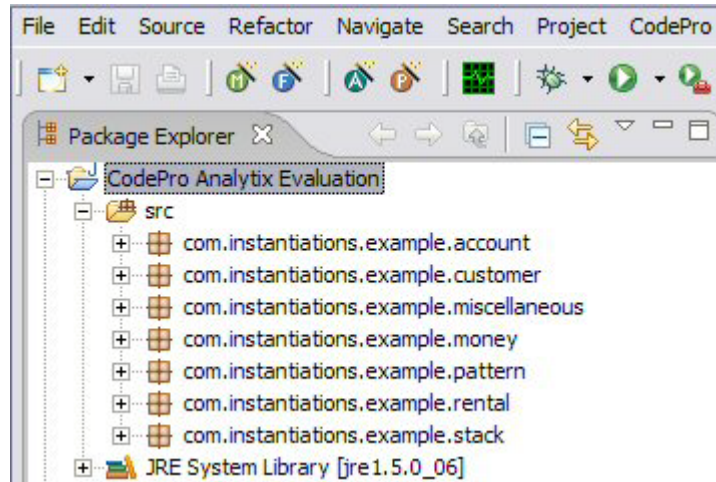
The CodePro AnalytiX evaluation package is a .zip file that contains a project that you will use in evaluating the Instantiations CodePro AnalytiX product. Do the following to start running the CodePro AnalytiX evaluation.

NOTE: Before installing CodePro AnalytiX, you must have an Eclipse-based IDE installed on your system.

1. First, install CodePro AnalytiX on your system. For Eclipse 3.4 and later based systems, you should install CodePro using our update site.
2. Once CodePro AnalytiX is installed on your system you'll want to create our CodePro evaluation project. You can do this by Choosing File > New > Other... from the menu or by hitting Control-N. Select the CodePro > CodePro Evaluation Project wizard.



3. Once the project is imported, you will see the following directory structure in the Package Explorer:



Follow the steps in the rest of the CodePro AnalytiX Evaluation Guide that describe how to perform tasks such as code audit, metrics, and collaboration.

How to Use this Guide

You can pick and choose which sections to run in this Evaluation Guide, but some refer back to skills described in earlier sections. In many of the exercises, you will follow the steps and not save your changes. If you accidentally save a change which modifies the project, you can simply re-create the Evaluation project (described in step#2 in the Getting Started section above) to return to a “default” project state.

Code Audit

The CodePro AnalytiX code audit feature checks your code against the built-in audit rules and determines areas where the code doesn't comply with those rules. The audit rules can be enabled or disabled and can be configured to work the way you want them to. The information about which rules are enabled and how they are configured is captured by an audit rule set. When non-compliant code is found, a violation is generated and the violations are gathered up into an audit result set.

CodePro AnalytiX includes over 1200 built-in audit rules, over 150 of which are targeted at finding Java security issues like SQL Injection and cross-site scripting, with more being added with every release.

CodePro allows for the exclusion of legacy code via date-based filtering. This filtering can be applied at the level of a single file, or at the level of the individual lines of code.

Manual Audit

Goals

In this section, you will learn how to use the manual code audit feature, which will allow you to select the portion of your code base to be audited and perform an audit using the default audit rule set. There are many times that you will want to manually audit your code, including:

- to verify a set of changes prior to checking them in to your revision control system,
- to validate one piece of code before building code that will depend on it, or
- to give you the peace of mind of knowing that you have not accidentally introduced a violation.

The manual audit feature lets you select the code to be audited and the audit rule set to be used. The violations that are found are displayed in the Audit view. In the Audit view, violations can be grouped by the rule that was violated, the rule's category, the rule's severity, the file containing the violating code, or even the author of the file containing the violation. In addition, CodePro AnalytiX displays over 400 Quick Fix hints providing detailed explanations on how to fix the violation.

Steps

1. In the *Package Explorer* view, right-click on the *CodePro AnalytiX Evaluation* project. On the context-menu, mouse over *CodePro Tools* to expand the sub-menu and then select *Audit Code*. This will open an *Audit* view which will show the code audit results.
2. The *Audit* view groups the results by audit rule by default. By clicking the buttons on the toolbar of the *Audit* view you can sort the results by rule, rule group, severity, resource, or author.

Click the button for *Group by Audit Rule Group*. 

3. In the *Audit* view, expand *Performance* and then expand *Append String [2]* to see the instances of that audit violation. Double click on the first instance of the violation (*MoneyBag.java – Line 122*) and the tool will open *MoneyBag.java* in the editor and take you to that specific violation.
4. In the *MoneyBag.java* editor, mouse-over the yellow colored flag in the gutter area next to the highlighted violation, a popup will tell you about the violation on that line.

5. Click on the flag in the gutter and you will see a quick-fix menu pop up with possible solutions for the audit violation. Double-click *Replace with character* and watch CodePro fix the violation. (The yellow flag in the gutter disappears).



```
buffer.append("{");
```

6. Scroll through the *MoneyBag.java* editor view and mouse over the flags in the gutter section to review the different types of errors and their associated quick-fix options. Close the *MoneyBag.java* editor when you are finished, but do not save any of the changes you have made.

Benefits

Developers can use the manual audit functionality in CodePro AnalytiX to easily find and fix code problems.

Using Different Rule Sets

Goals

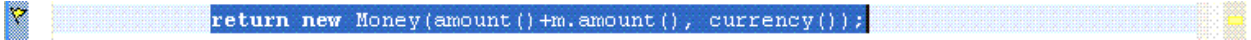
CodePro allows you to define different audit rule sets for different purposes, but only one rule set can be selected as the default at any given time. In this section you, will learn how to select an audit rule set other than the default when using the manual code audit feature. Although not shown here, you can also use multiple audit rule sets at the same time.

Features

CodePro AnalytiX allows you to use different audit rule sets for different purposes. CodePro AnalytiX ships with seven pre-defined audit rule sets and allows you to define your own.

Steps

1. In the *Package Explorer* view, right-click on the *CodePro AnalytiX Evaluation* project. On the context-menu, mouse over *CodePro Tools* to expand the sub-menu and then select *Audit Code Using...*
2. In the *Choose Audit Rule Sets* dialog, de-select (uncheck the box) *CodePro Default* and select (check the box) *Potential Errors and Refactorings* then click *OK*. Wait while the *Audit* view is updated with the new audit results. **Note:** multiple audit rule sets can be selected in this dialog.
3. In the *Audit* view, expand *Coding Style* and then expand the *Missing Block [12]* violation list (you may need to click the *Group by Audit Rule* button if your violations are still grouped by *Audit Rule Group*). Double-click on the first violation (*Missing block in if clause: return new Money(amount()+m.amount(), currency()); (Money.java - Line 26)*). This will open the *Money.java* file in an editor view and take you to the line where the violation occurs.



4. In the *Money.java* editor, click the yellow colored flag in the gutter area next to the highlighted violation, a popup will provide a list of possible quick-fixes for the violation.
5. Double-click *Change 'if' statement to block* and watch CodePro fix the violation. Close the *Money.java* editor view when you are finished, but do not save any of the changes you have made.

Benefits

It is easy to select the right audit rule set in CodePro AnalytiX so that code is being audited using the most appropriate audit rules.

Choosing and Importing Rule Sets

Goals

In this section, you will learn how to import an audit rule set and make it be the default audit rule set. This is useful to allow you to share audit rule sets from other people, such as an audit rule set representing your development group's coding standards.

Features

CodePro AnalytiX lets you choose the audit rules that you want to use, including rule sets used by your team that are not part of CodePro AnalytiX. You can also select the audit rule set to use as the default rule set.

Steps

1. Open the CodePro Audit preferences by selecting *Preferences > Audit* from the *CodePro* menu.
2. Click on the *Rule Sets* tab to see a list of available rule sets.
3. Import a new rule set by clicking the *Import* button. Browse to the *CodePro AnalytiX Evaluation* folder in your workspace, select the *Project* tab and select *EvalGuideRules.pref* then click *OK*.
4. Notice that the *Eval Guide Rules* rule set is now listed and marked as default with an asterisk. Click the *OK* button to exit the preferences dialog.
5. Right-click on the *CodePro Evaluation* project and select the menu item *CodePro Tools > Audit Code* from the context menu to audit the code using the new rule set.

Benefits

You can set the audit rule sets that are used in the tool and even select the audit rule set to use as the default rule set.

Dynamic Audit

Goals

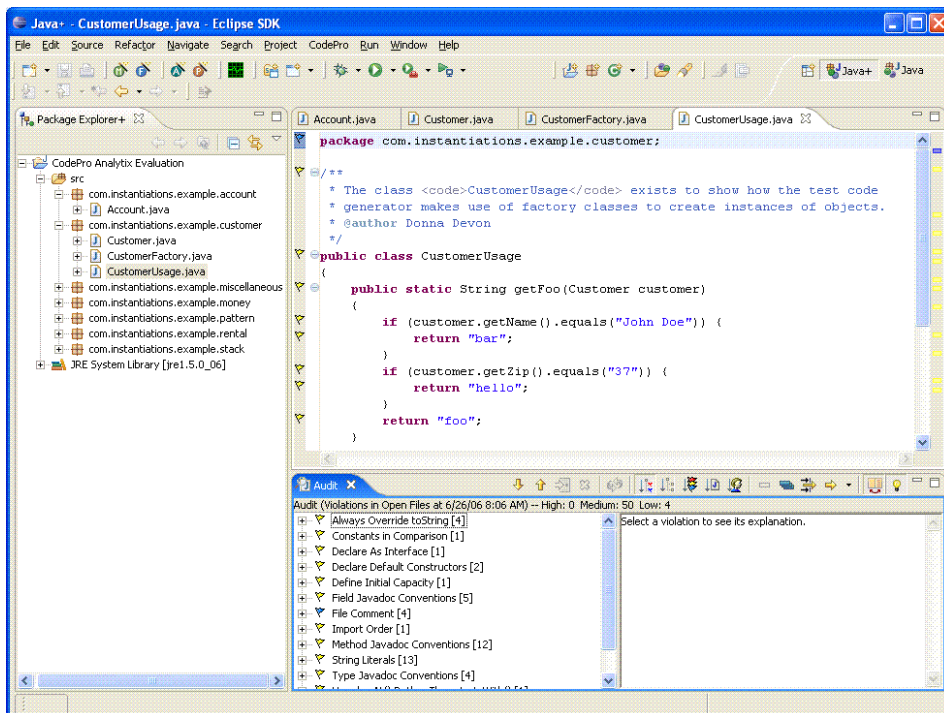
In this section, you will learn how to dynamically audit every file as it is opened or saved. Auditing files while they are open in an editor will help you catch potential problems even sooner, decreasing the cost of fixing those problems. Although not shown here, you can also create filters so that only a subset of the opened files, such as those in your projects, are audited.

Features

The CodePro AnalytiX code audit function can be set to run dynamically, in which case the tool will audit the code in all of the currently open files, updating the analysis every time you open, close or save a file. This allows you to catch violations as you're writing the code. For example, a single naming violation might be propagated through dozens of source files before the nightly build's audit would flag the problem. Dynamic audit would flag the naming violation so that it can be corrected as soon as the violation occurs. With dynamic code audit turned on, you will actually see colored flags indicating errors appear and disappear throughout the course of the day as you introduce or correct errors.

Steps

1. Open the CodePro Audit preferences by selecting *Preferences > Audit* from the *CodePro* menu.
2. Enable dynamic code auditing by clicking on the *Dynamic* tab and checking the box for *Dynamically audit code*. Choose *CodePro Default* from the *Use the audit rule set:* pulldown menu and then click *OK* to exit the preferences dialog.
3. In the *Package Explorer*, expand the *CodePro AnalytiX Evaluation* project and open/close editor views of different classes from within the project. Watch the *Audit* view as the audit violations appear and disappear (increase/decrease in count).



4. Turn off *Dynamically audit code* in the CodePro Audit preferences when you are finished and close any open Editor views.

Benefits

Dynamic auditing saves developers time and results in higher quality code. Dynamic code audit executes continuously so developers spend less time in code reviews and more time coding.

Custom Rule Set Generation

Goals

In this section, you will learn how to define your own audit rule set and export it so that it can be shared with other developers. This is useful, for example, if you are responsible for creating and distributing an audit rule set representing your development group's coding standards.

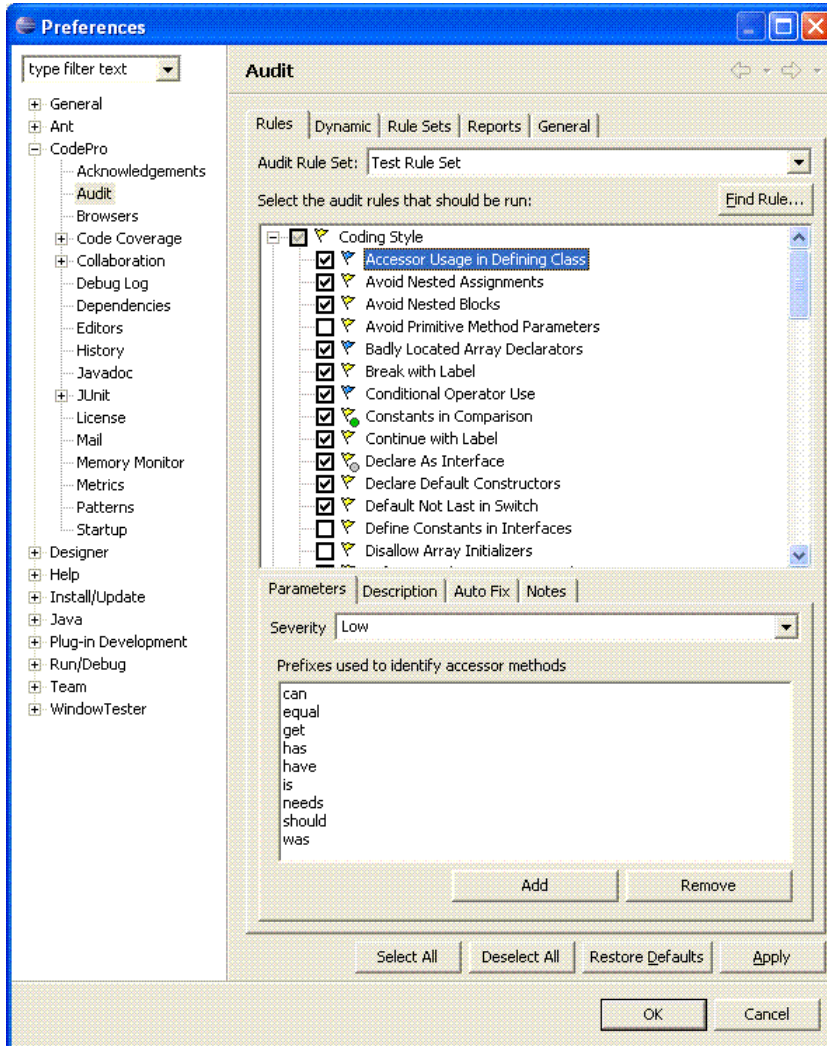
Features

CodePro AnalytiX audit rule sets are flexible—it is easy to choose which audit rules will be used and to customize those audit rules to reflect your best practices coding standards. You can also modify existing audit rule sets (unless they are locked) by turning rules on and off, setting the level of severity, or modifying the behavior of the audit rules. Most of the audit rules can be customized to further fine tune the behavior of a specific rule.

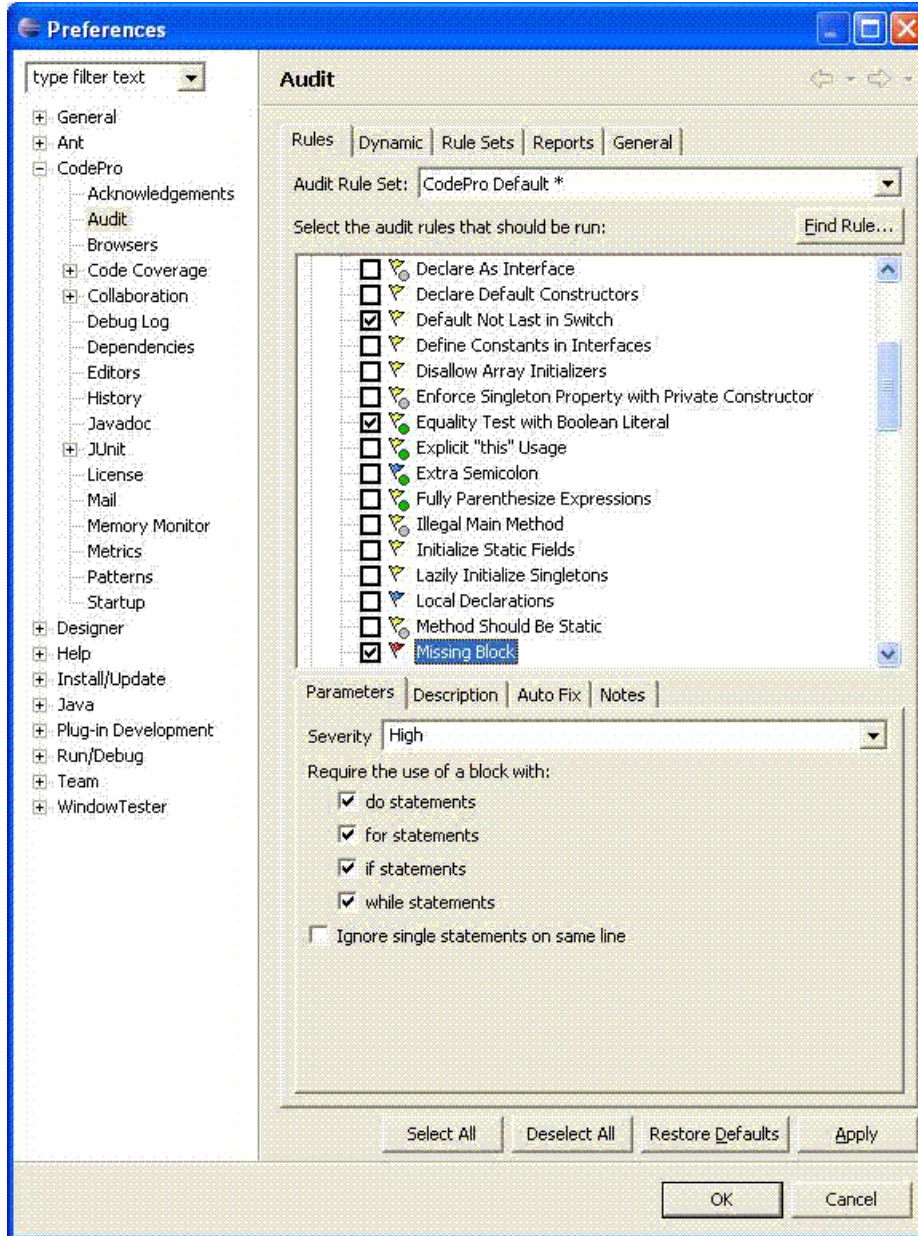
Steps

1. Open the CodePro Audit preferences by selecting *Preferences > Audit* from the *CodePro* menu.
2. Create a new rule set by clicking on the *Rule Sets* tab and clicking the *New* button. Type in a name for your rule set (such as Test Rule Set) and then click *Finish*.
3. To begin editing your new rule set, click on the *Rules* tab and verify that your new rule set is selected in the *Audit Rule Set* pulldown menu.

4. Begin adding rules to your new rule set by expanding the *Coding Style* rule group in the tree and selecting the first rule (*Accessor Usage in Defining Class*). Check the box next to the rule and change the severity of this violation to *Low* (on the *Parameters* tab below) as shown in the following figure.



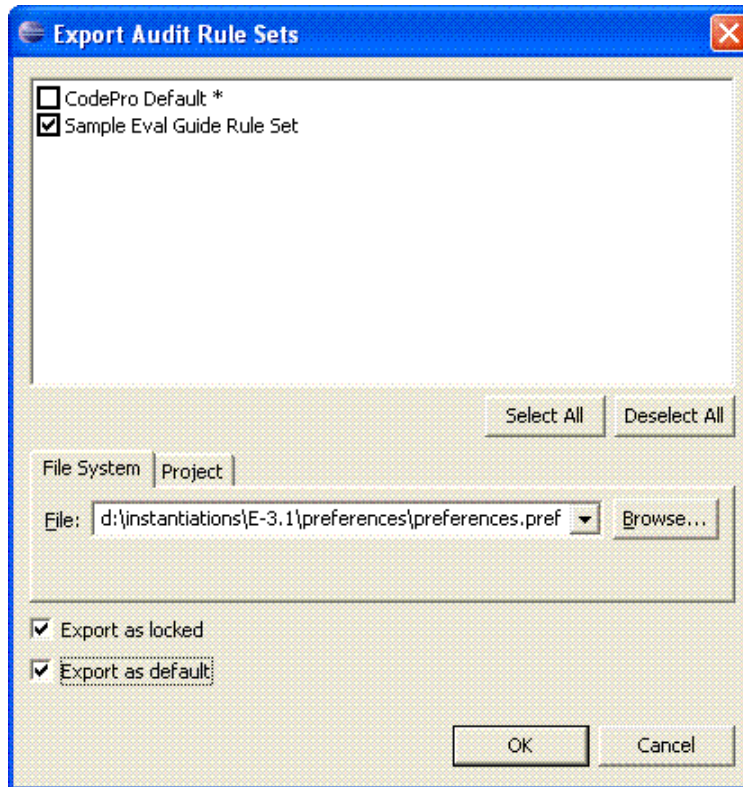
- Click the *Find Rule* button and begin typing 'missing block' until you see it in the *Matching audit rules* list. Double click on the *missing block* rule to jump to that rule on the *Rules* tab. Make sure the box is checked to turn on the rule and change the severity to *High* on the *Parameters* tab. Also de-select the box labeled 'Ignore single statements on same line' as shown in the following figure.



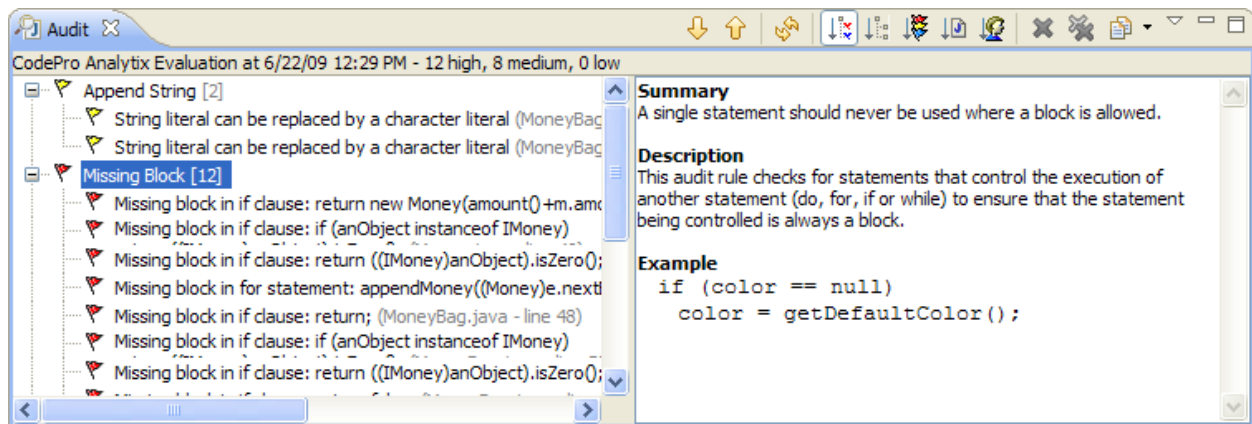
- Update the notes for this rule by clicking on the *Notes* tab and adding a comment.
- Save your changes to this rule set by clicking the *Apply* button.
- Set this new rule set as your default by selecting the *Rule Sets* tab and clicking the *Set Default* button (note that the selected rule set now has an asterisk next to it indicating it is the default rule set). Click the *Apply* button to make this change take effect.

CodePro AnalytiX Evaluation Guide

- Export this rule set by clicking the *Export* button. In the *Export Audit Rule Sets* dialog make sure the new rule set is marked for export. Choose a location for the exported rule set by clicking the *Browse* button (or typing the path and filename directly). Check the boxes for *Export as locked* and *Export as default* and then click *OK* to exit the dialog.



- Run an audit of the *CodePro AnalytiX Evaluation* project using the default rule set. Note in the Audit view that the *Missing Block* violations are now marked with a red (high) priority flag.



- Close the Audit view and reconfigure the *CodePro Default* rule set as your default in the Audit preferences.

Benefits

CodePro AnalytiX makes it easy to add and customize audit rules so they incorporate your own unique audit rules. These rules then become part of the CodePro AnalytiX audit rule sets. You can also share these rules.

Using Audit Reports

Goals

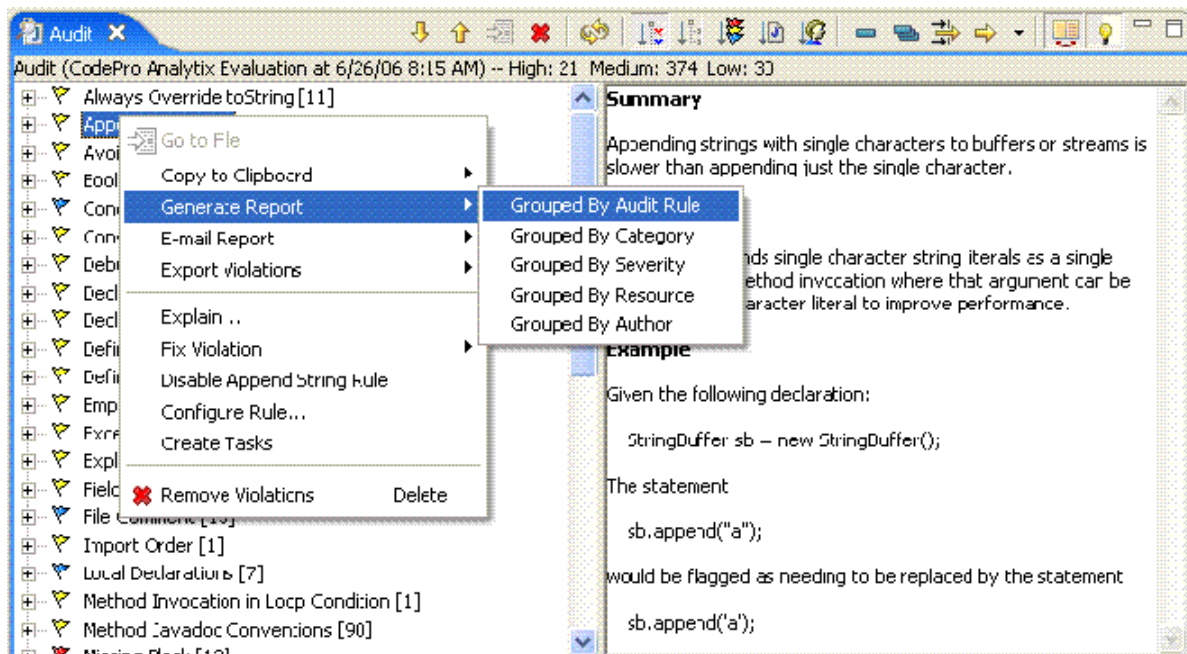
In this section, you will learn how to generate an HTML report containing the results of a code audit. This is useful for reporting to management on the status of a project.

Features

CodePro AnalytiX can generate a full range of reports in HTML, XML and text format that indicate what the tool found during a code audit. From the Audit view you can generate reports on the audit in different report formats; plain text, tab or comma delimited for importing into a spreadsheet, or in HTML. Reports can be sorted in a number of manners, including by rule, or category, or severity. In addition, it is possible to generate e-mail and send individual reports to selected users or managers.

Steps

1. Invoke a code audit of the *CodePro AnalytiX Evaluation* project.
2. In the *Audit* view, right-click on one of the audit violations and open the *Generate Report* sub-menu and select *Grouped by Audit Rule*.



CodePro AnalytiX Evaluation Guide

3. Enter a name for the report (such as Example Audit Report by Rule) and click Save (be sure to note the save location or change it to something easily accessible).
4. Browse to the location where you saved the report and double-click the file to open it in a web browser.

CodePro Code Audit for CodePro AnalytiX Evaluation
 This document contains the results of auditing CodePro AnalytiX Evaluation at 6/22/09 11:18 AM, using the default audit rule set.

Violation Counts by Severity

Violation Severity	Violation Count
High	0
Medium	8
Low	0

Violations by Audit Rule

Audit Rule	Count
Append String	2
Non-case Label in Switch	1
Obey General Contract of Equals	2
String Comparison	1
Unused Label	1
Use charAt() Rather Than startsWith()	1

Append String (2) [\[top\]](#)

Violation	Recommendation	Severity	Resource	Line
String literal can be replaced by a character literal	Replace the string literal with a character literal.	Medium	MoneyBag.java	122
String literal can be replaced by a character literal	Replace the string literal with a character literal.	Medium	MoneyBag.java	125

Non-case Label in Switch (1) [\[top\]](#)

Violation	Recommendation	Severity	Resource	Line
Non-case label in switch: case10	Add the keyword "case" or remove the label.	Medium	Samples.java	76

Obey General Contract of Equals (2) [\[top\]](#)

Violation	Recommendation	Severity	Resource	Line
Missing identity check	Add a test for object identity.	Medium	Money.java	38
Missing identity check	Add a test for object identity.	Medium	MoneyBag.java	51

String Comparison (1) [\[top\]](#)

Violation	Recommendation	Severity	Resource	Line
Cannot compare strings using the equals (==) operator	Replace the comparison with equals().	Medium	Samples.java	30

Unused Label (1) [\[top\]](#)

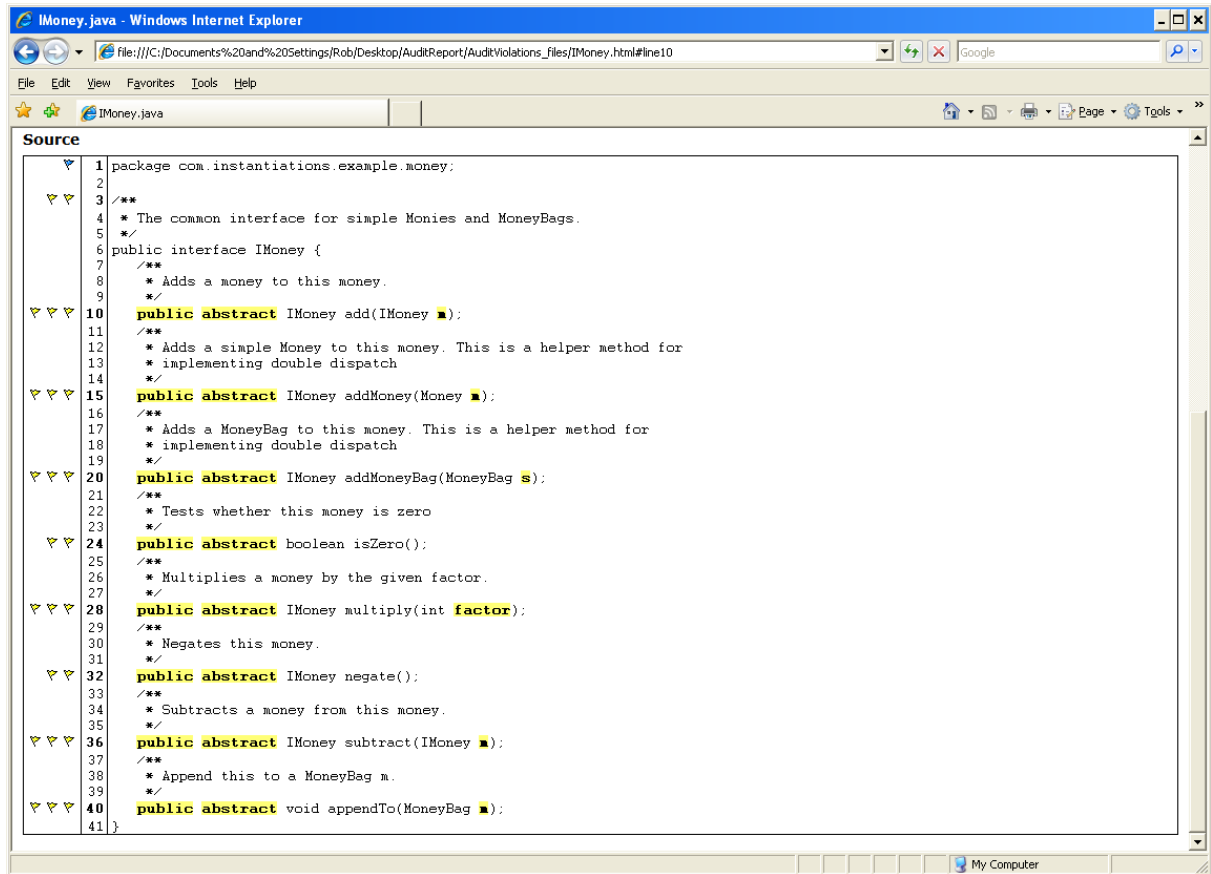
Violation	Recommendation	Severity	Resource	Line
-----------	----------------	----------	----------	------

CodePro AnalytiX Evaluation Guide

- Click on any of the summarized audit violations (e.g Obsolete Modifier Usage) at the top of the page to jump to those violations in the report. Note that for each audit violation, the report includes a description, a recommendation, the severity, and the location by file and line as shown in the following figure.

Violation Type	Recommendation	Severity	Resource	Line
Obsolete Modifier Usage [16] top				
Use of obsolete modifier: public	Remove the obsolete modifier.	Medium	IMoney.java	10
Use of obsolete modifier: abstract	Remove the obsolete modifier.	Medium	IMoney.java	10
Use of obsolete modifier: public	Remove the obsolete modifier.	Medium	IMoney.java	15
Use of obsolete modifier: abstract	Remove the obsolete modifier.	Medium	IMoney.java	15
Use of obsolete modifier: public	Remove the obsolete modifier.	Medium	IMoney.java	20
Use of obsolete modifier: abstract	Remove the obsolete modifier.	Medium	IMoney.java	20
Use of obsolete modifier: public	Remove the obsolete modifier.	Medium	IMoney.java	24
Use of obsolete modifier: abstract	Remove the obsolete modifier.	Medium	IMoney.java	24
Use of obsolete modifier: public	Remove the obsolete modifier.	Medium	IMoney.java	28
Use of obsolete modifier: abstract	Remove the obsolete modifier.	Medium	IMoney.java	28
Use of obsolete modifier: public	Remove the obsolete modifier.	Medium	IMoney.java	32
Use of obsolete modifier: abstract	Remove the obsolete modifier.	Medium	IMoney.java	32
Use of obsolete modifier: public	Remove the obsolete modifier.	Medium	IMoney.java	36
Use of obsolete modifier: abstract	Remove the obsolete modifier.	Medium	IMoney.java	36
Use of obsolete modifier: public	Remove the obsolete modifier.	Medium	IMoney.java	40
Use of obsolete modifier: abstract	Remove the obsolete modifier.	Medium	IMoney.java	40
Questionable Assignment [1] top				
Questionable method parameter assignment: arg = new String(arg)	Check the logic of the assignment statement.	Low	Samples.java	26
Questionable Name [7] top				
Questionable name found: foo	Rename.	High	Samples.java	13
Questionable name found: bar	Rename.	High	Samples.java	24
Short name found: m	Rename.	High	MoneyBag.java	62
Short name found: m	Rename.	High	MoneyBag.java	72
Short name found: m	Rename.	High	MoneyBag.java	86
Short name found: m	Rename.	High	MoneyBag.java	98
Short name found: m	Rename.	High	MoneyBag.java	107
Spell Check Identifiers [15] top				
The string "zip" is not a word.	...	Low	Customer.java	17
The string "zip" is not a word.	...	Low	Customer.java	21
The string "Zip" is not a word.	...	Low	Customer.java	32
The string "iane" is not a word.	...	Low	CustomerFactory.java	15
The string "Foo" is not a word.	...	Low	CustomerUsage.java	10
The string "foo" is not a word.	...	Low	Samples.java	13
The string "arg" is not a word.	...	Low	Samples.java	13
The string "arg" is not a word.	...	Low	Samples.java	24
The string "xyz" is not a word.	Adds the word 'xyz' to the dictionary	Low	Samples.java	38

- The itemized list of violations can then take you to the source in which the individual violation is found by clicking on the specific violation.



- Close the report when you are finished.

Benefits

The CodePro AnalytiX report feature is a valuable tool in tracking the problems CodePro located in your code and in providing information that can be used for analysis and troubleshooting.

Similar Code Analysis

Identifying Similar Code

Goals

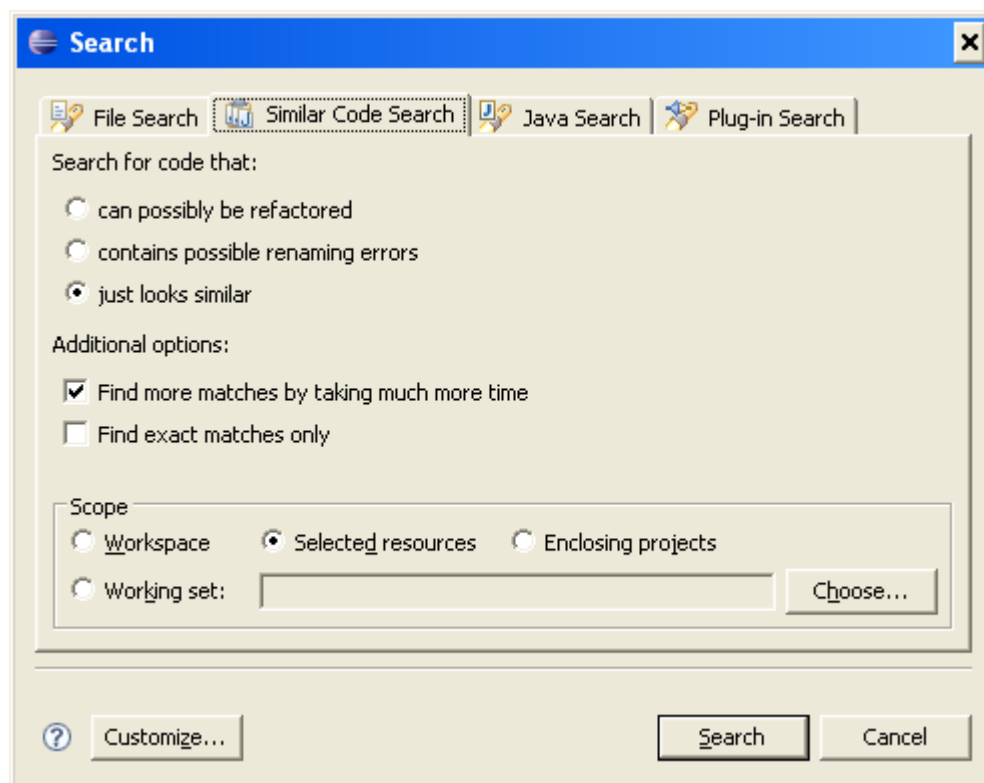
In this section, you will learn how to perform similar code analysis. This makes it very easy for developers to identify copied fragments of code throughout a project and easily go back and refactor them.

Features

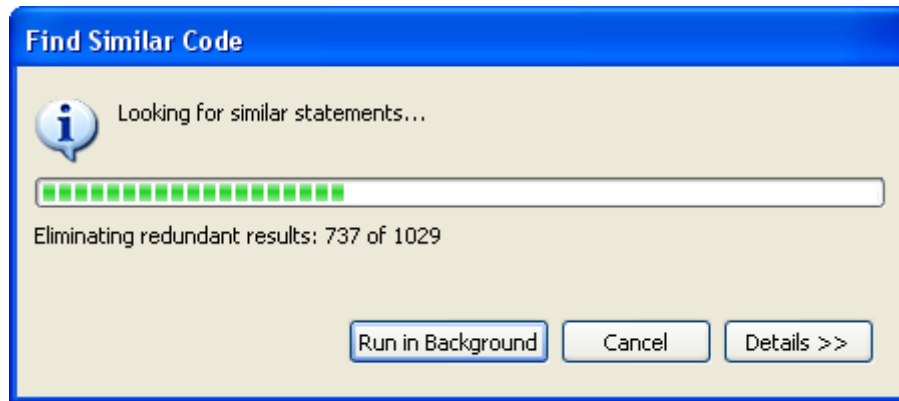
The CodePro AnalytiX similar code analysis facility provides developers with the capability to easily find and identify copied bits of code, very similar fragments of code, or even copied code with renamed or manipulated variables. This makes refactoring code much faster and more efficient.

Steps

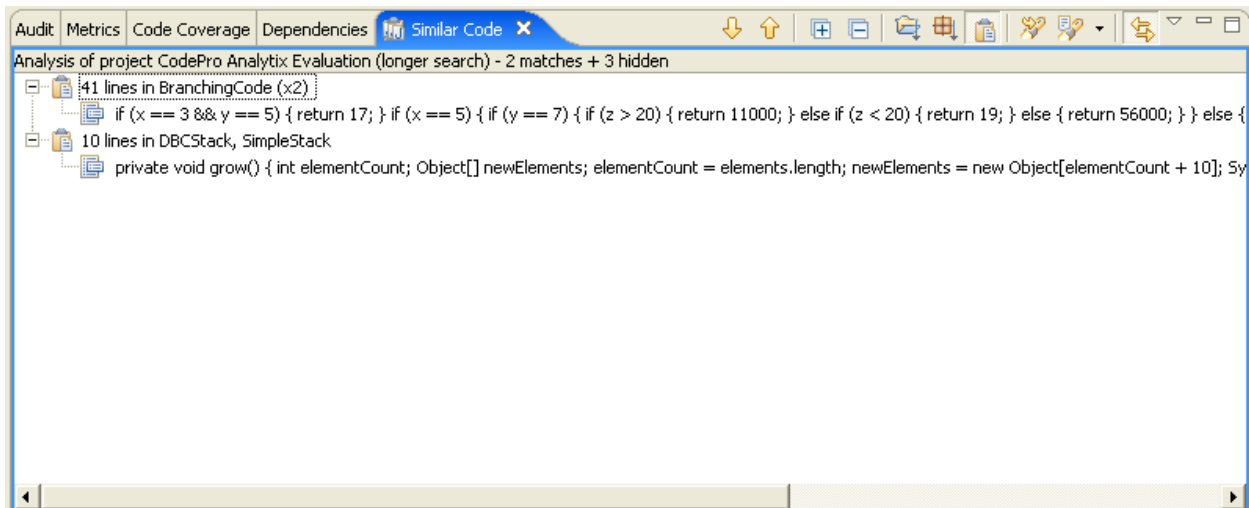
1. Select the CodePro AnalytiX Evaluation project in either the *Navigator* or *Project Explorer* view, and select *SimilarCode...* from the *Search* menu. You'll get the following dialog box:



2. Make sure the *Find more matches by taking much more time* checkbox is checked and choose *Selected resources* as the scope. Click the *Search* button. Operation progress will be displayed.



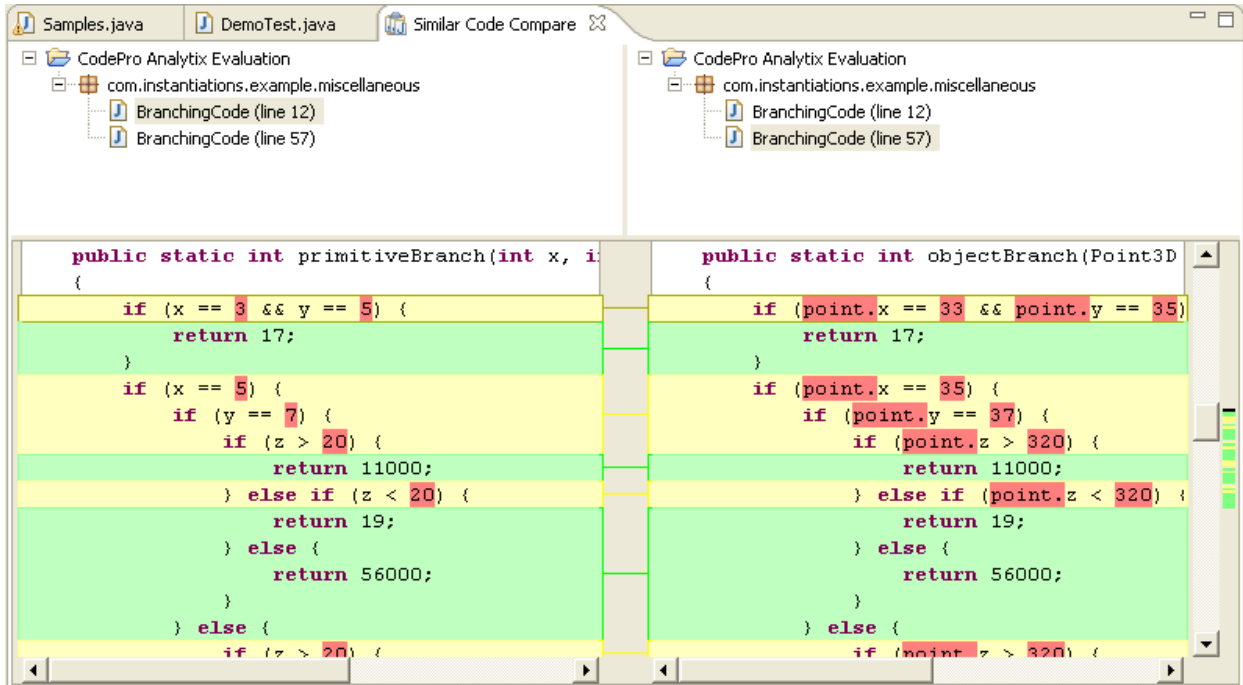
3. The *Similar Code* view opens automatically when the analysis is complete. The list of matches is displayed. (Note: If the list is empty, make sure you've set the checkbox in the previous step).



4. Use the *Group By...* buttons to switch between three grouping modes and choose the one you like most (Project, Package or Match).



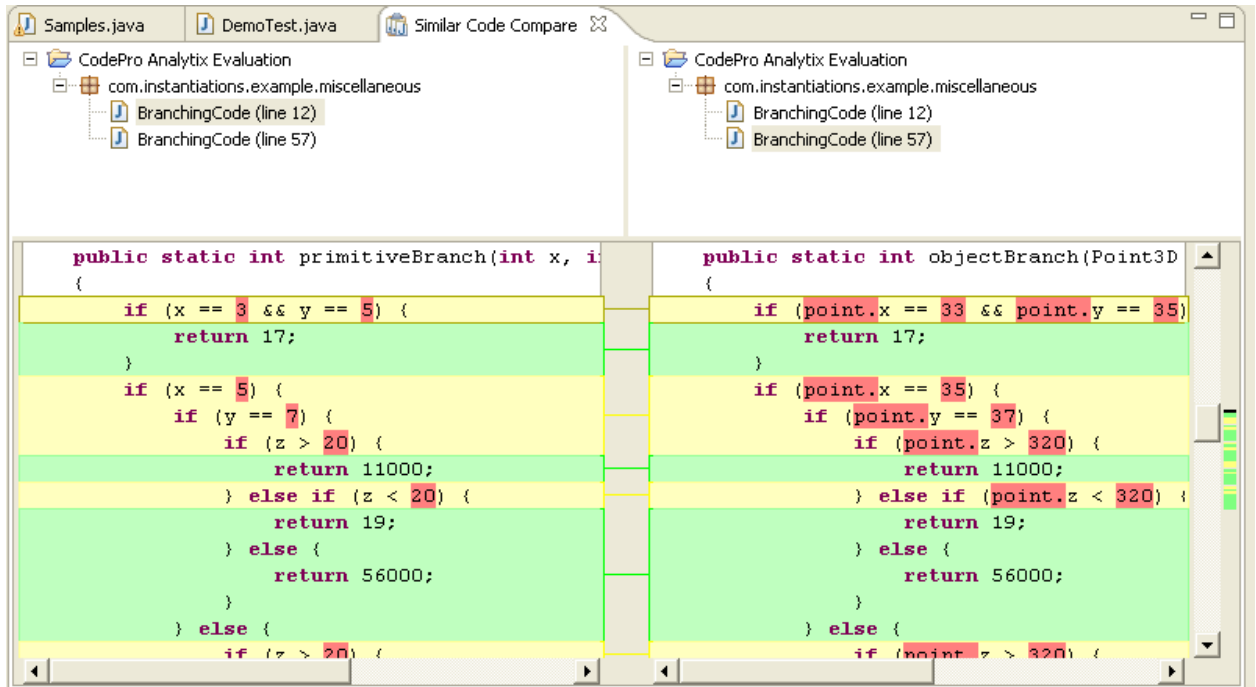
- Click on a match to open the *Compare Editor*.



- When the *Similar Code* view is active, you can navigate between matches by using the *Next Match* and *Previous Match* buttons, or by pushing SPACE and SHIFT SPACE on the keyboard.



- The editor shows textual differences between the matched code snippets. Green lines designate identical code, yellow lines designate differing code (with differing tokens highlighted with a read background), and red lines designate inserted or removed code.



Benefits

The CodePro AnalytiX similar code analysis helps you find pieces of code that are similar. Similar code can often be refactored into separate methods so that it can be reused, resulting in a smaller code base that is easier to test and maintain. Even when the code cannot be refactored, the similar code analysis can often help you find bugs caused by copying code and making incomplete changes.

Metrics

Computing Metrics

Goals

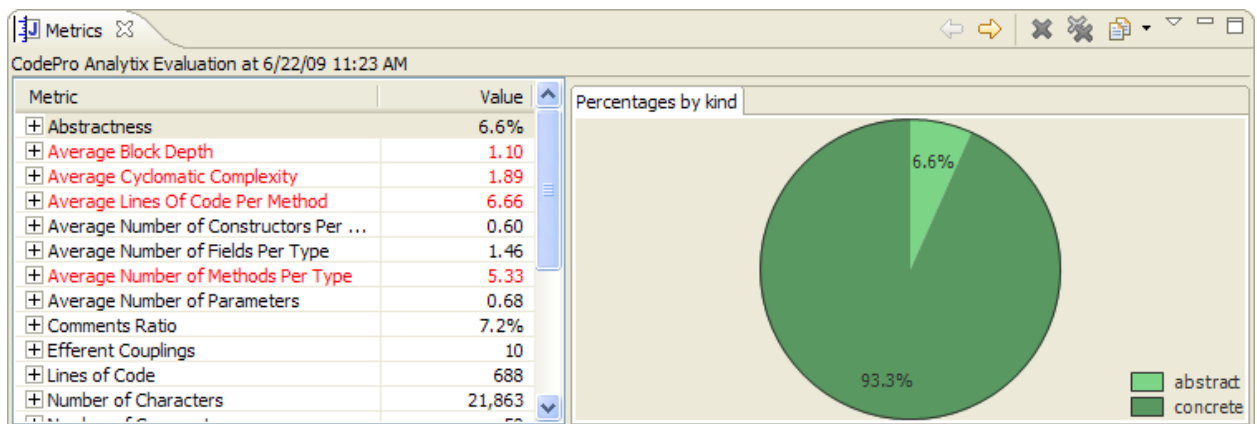
In this section, you will learn how to compute metrics for your code. Metrics are another means of understanding the quality of a code base.

Features

The CodePro AnalytiX metrics facility can provide managers or developers with vital information about code. Reports can also be exported into an HTML report that can be distributed to project managers, supervisors or developers.

Steps

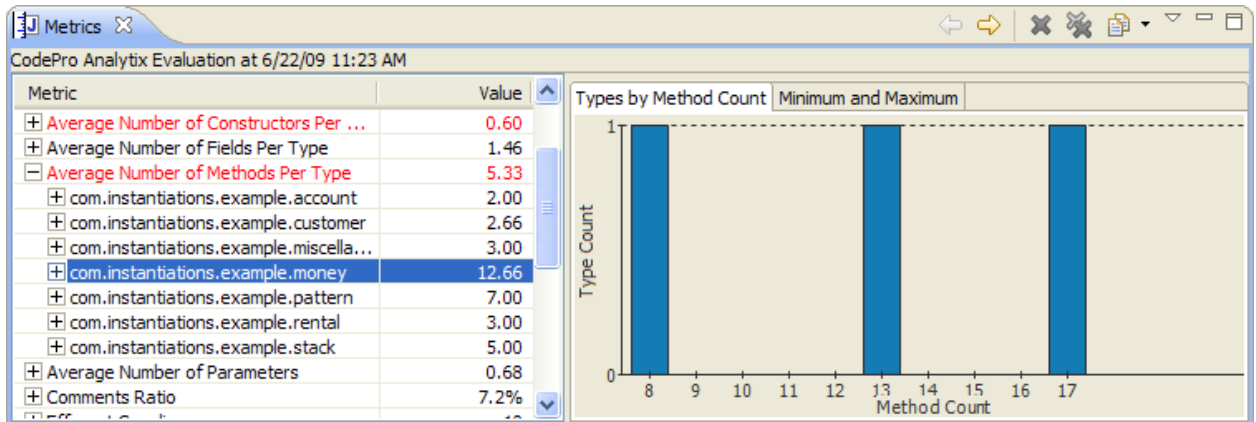
1. In the *Package Explorer* view, right-click on the *CodePro AnalytiX Evaluation* project. On the context-menu, mouse over *CodePro Tools* to expand the sub-menu and then select *Compute Metrics*. This will open a *Metrics* view which will show the computed metrics (if the view opens but the metrics do not display, re-select *Compute Metrics* from the *CodePro Tools* sub-menu).
2. In the *Metrics* view, you'll see the project level computed metrics and those metrics that exceeded the default threshold will be highlighted with red text. Selecting individual metrics will update the right-hand frame with additional details or graphs of the selected metric. Click on *Abstractness* to see the associated chart.



3. Click through the various metrics to see the associated charts and tables in the right-hand frame.
4. Select the *Average Number of Methods Per Type* and note that the right-hand frame updates to show the types by method count. Choose the *Minimum and Maximum* tab to see the bounds of this metric.

CodePro AnalytiX Evaluation Guide

5. In the left-hand frame, expand the *Average Number of Methods Per Type* and select the package *com.instantiations.example.money*, right-click on the package and select *Go Into*. This recalculates the top-level metrics from this specific package level. In the following figure, note that the left-hand frame updates to reflect these new metrics.



6. Click the 'back arrow' in the *Metrics* view to return to the project level metrics, then close the *Metrics* view to exit the metrics results.

Benefits

The CodePro AnalytiX metrics tool is valuable in providing information on code, projects and performance.

Choosing and Creating Metric Sets

Goals

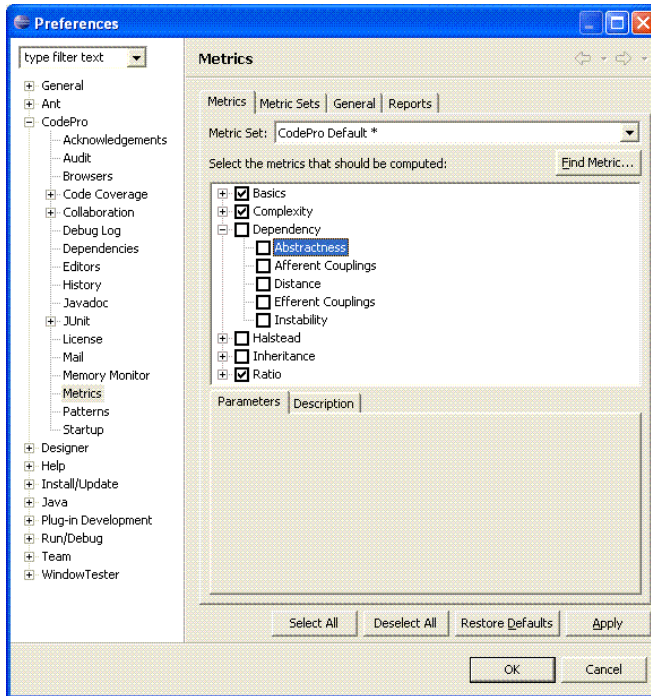
In this section, you will learn how to create a new metric set and how to make it the default metric set. This is useful, for example, if you are responsible for creating and distributing a metric set representing your development group's standards.

Features

Developers use the CodePro AnalytiX Metric facility to configure metrics in a variety of categories including basics, complexity, dependency, Halstead, inheritance and ratio categories. When developers select Metrics, the enabled metrics in the default metric set are run and displayed in a table. The individual metrics may be expanded (drill down) to show the value of that metric for project, package and type. Any metrics that have exceeded their user-defined thresholds and trigger points are shown in a highlighted color.

Steps

1. From the *CodePro* menu, select *CodePro > Preferences > Metrics* to go to the *Metric* preferences then click on the *Metric Sets* tab. Click the *New* button and enter a name for your new metric set (such as *Eval Guide Metric Set*) and then click *OK*.
2. Click on the *Metrics* tab and make sure that the *Metric Set* pull-down menu is set to the new set you created. Expand the *Dependency* group and select *Abstractness*. Click on the *Description* tab to see a detailed calculation of the abstractness metric. After reading the description de-select *Abstractness* from the list of metrics. Click on the *Efferent Couplings* metric, read the description and check the *Parameters* tab before de-selecting it.



3. Check the box next to *Halstead* metrics and then expand the list. Observe that all of the subsequent metrics have been selected as well. Click through the selected metrics and read their associated descriptions and parameters. When you are finished, click the *Apply* button and then click *OK*.
4. In the *Package Explorer*, right-click on the *CodePro AnalytiX Evaluation* project and select *CodePro Tools* and then select *Compute Metrics Using...* In the *Selection Needed* dialog box, de-select *CodePro Default* and select your custom metric set then click *OK*.
5. Review the results of your newly computed metrics in the *Metrics* view before closing the view.

Benefits

Developers can set metrics in a variety of categories letting them customize the tool to match site requirements.

Metric Reports

Goals

In this section, you will learn how to generate an HTML report containing the results of a metric computation. This is useful for reporting to management on the status of a project.

Features

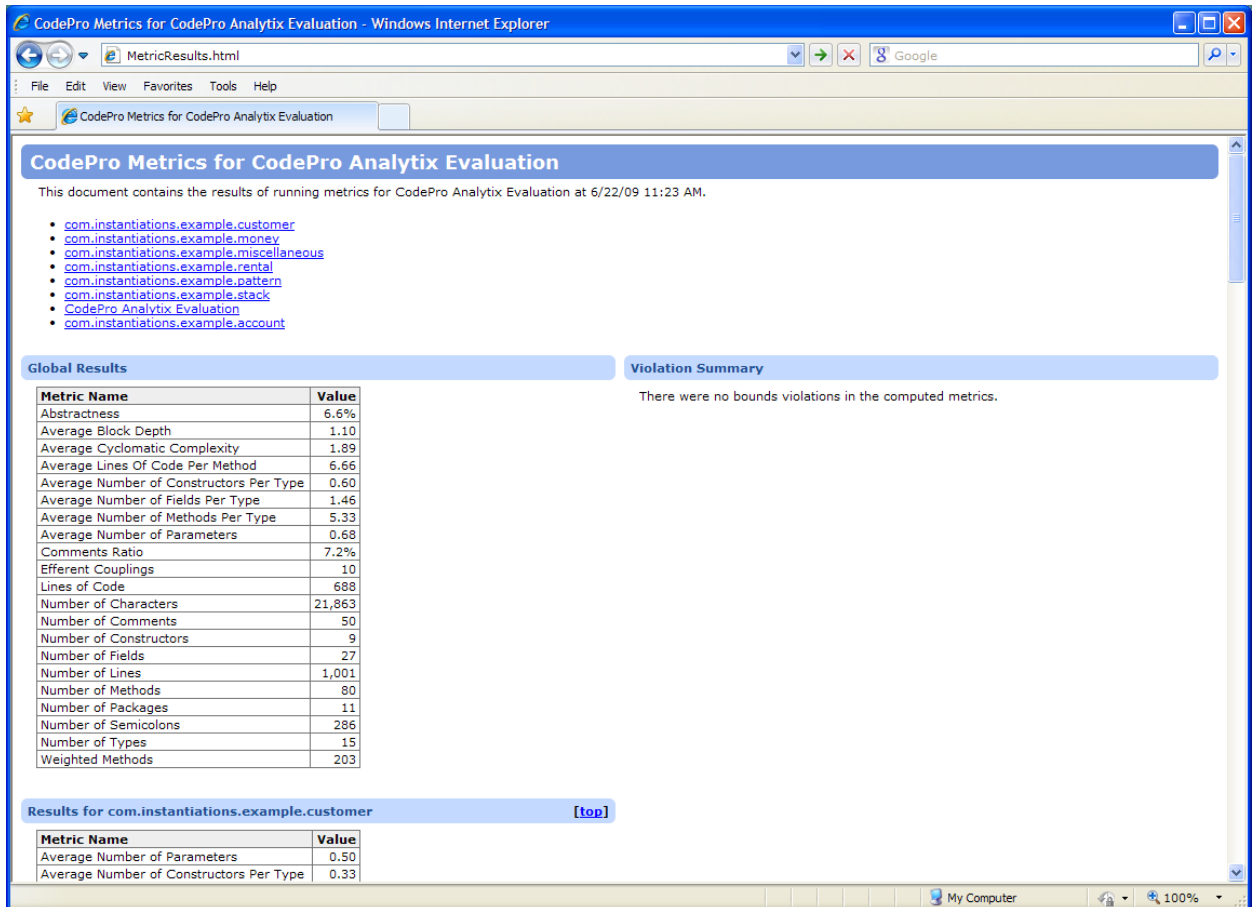
CodePro AnalytiX can display metric information in a variety of formats including various reports, tables, charts and graphs. Metric results can be captured to the clipboard so that they can be pasted into e-mail messages, text documents, HTML documents, XML documents or even a spreadsheet application.

Steps

1. Compute metrics on a project in your workspace (e.g. *CodePro AnalytiX Evaluation*).
2. In the *Metrics* view, right-click on a metric and select *Export Results* and then select *As HTML*. In the dialog box, specify a name and location for your metrics report (such as Eval Guide Metric Report) and then click *OK*.

CodePro AnalytiX Evaluation Guide

- When the export is complete, browse to the file location of your metric report and double-click the file to open it in a web browser.



- Notice that the metric report includes project level metrics at the top of the page and also includes package-specific metrics for each package in your project. When you finish reviewing your metric report, close the web browser and return to your IDE.

Benefits

A variety of metric reports can be created in CodePro AnalytiX and metric information can easily be shared across teams or locations.

JUnit Test Generation

Generating Test Cases

Goals

In this section, you will learn how to generate JUnit tests for the classes in your projects. Having unit tests can help you more quickly identify regressions in the functionality of your code.

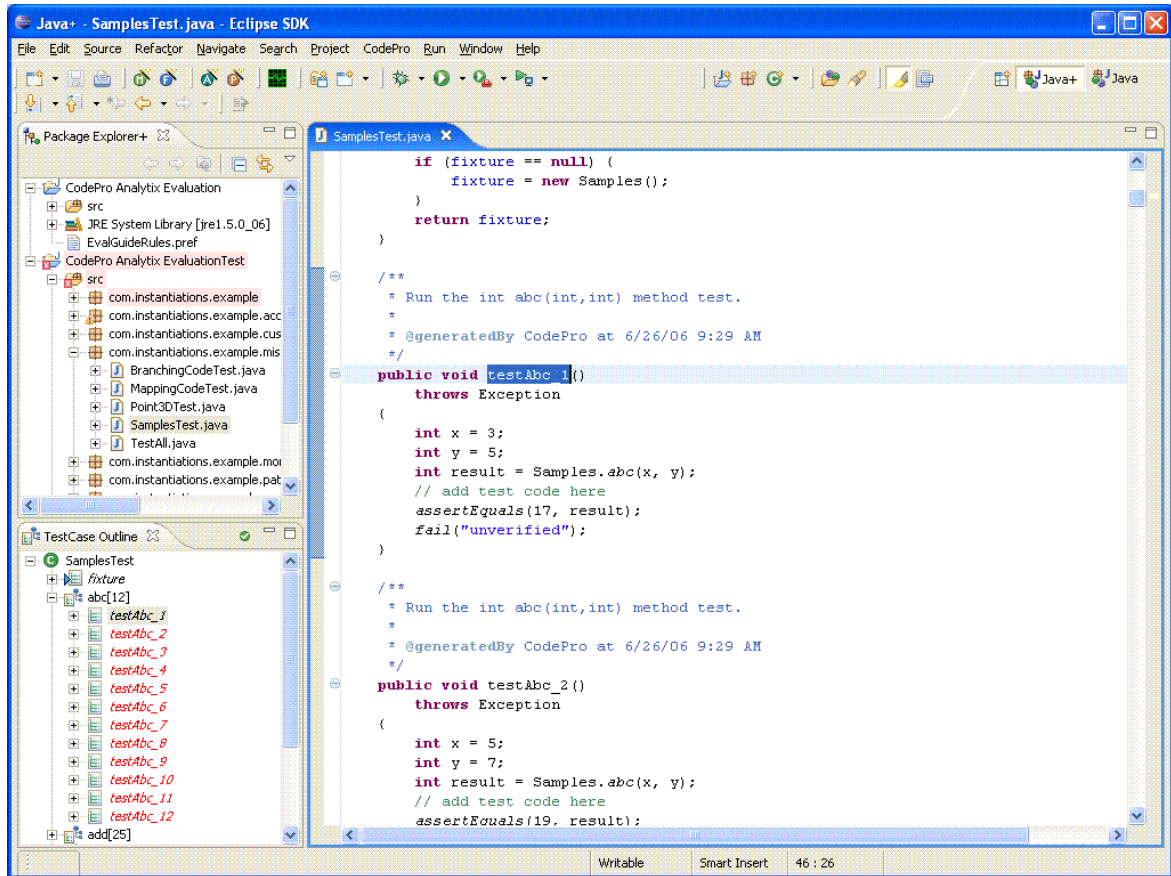
Features

The CodePro JUnit Test Case Generation facility helps developers automate the creation of comprehensive JUnit test cases. It does this through a combination of both static code analysis and by dynamically executing the code to be tested in order to observe the behavior of the code.

Steps

1. In the *Package Explorer* view, right-click on the *CodePro AnalytiX Evaluation* project. On the context-menu, mouse over *CodePro Tools* to expand the sub-menu and then select *Generate Test Cases*. When the test generation has completed, you will see a new project called *CodePro AnalytiX EvaluationTest* in the *Package Explorer*.
2. From the *CodePro* menu, expand the *Views* submenu and select *TestCase Outline*.
3. In the *Package Explorer*, expand the *CodePro AnalytiX EvaluationTest* project and the *src* folder, then expand *com.instantiations.example.miscellaneous* and double-click the *SamplesTest.java* file to open it. Notice in the *TestCase Outline* view that it updates to reflect the tests in the opened file.

4. In the *TestCase Outline* view, expand *Abc* in the hierarchy and select *testAbc_1*, (notice that the editor view jumps to that section), and review the generated test case.



5. After reviewing the test case, right-click on it in the *TestCase Outline* view and select *Mark Verified*. You can repeat this for all of the generated test cases or move on to the next step.
6. When your tests have been verified, you can run them by right-clicking on *TestAll.java* in the *Package Explorer* (under *CodePro AnalytiX EvaluationTest* -> *src* -> *com.instantiations.example*) and selecting the *Run As* sub-menu and then select *JUnit Test*.

Note: any unverified tests will be marked 'fail' in execution until verified.

Benefits

CodePro AnalytiX generates JUnit tests that developers can use to help maintain the highest quality in their code.

Generating Factory Classes

Goals

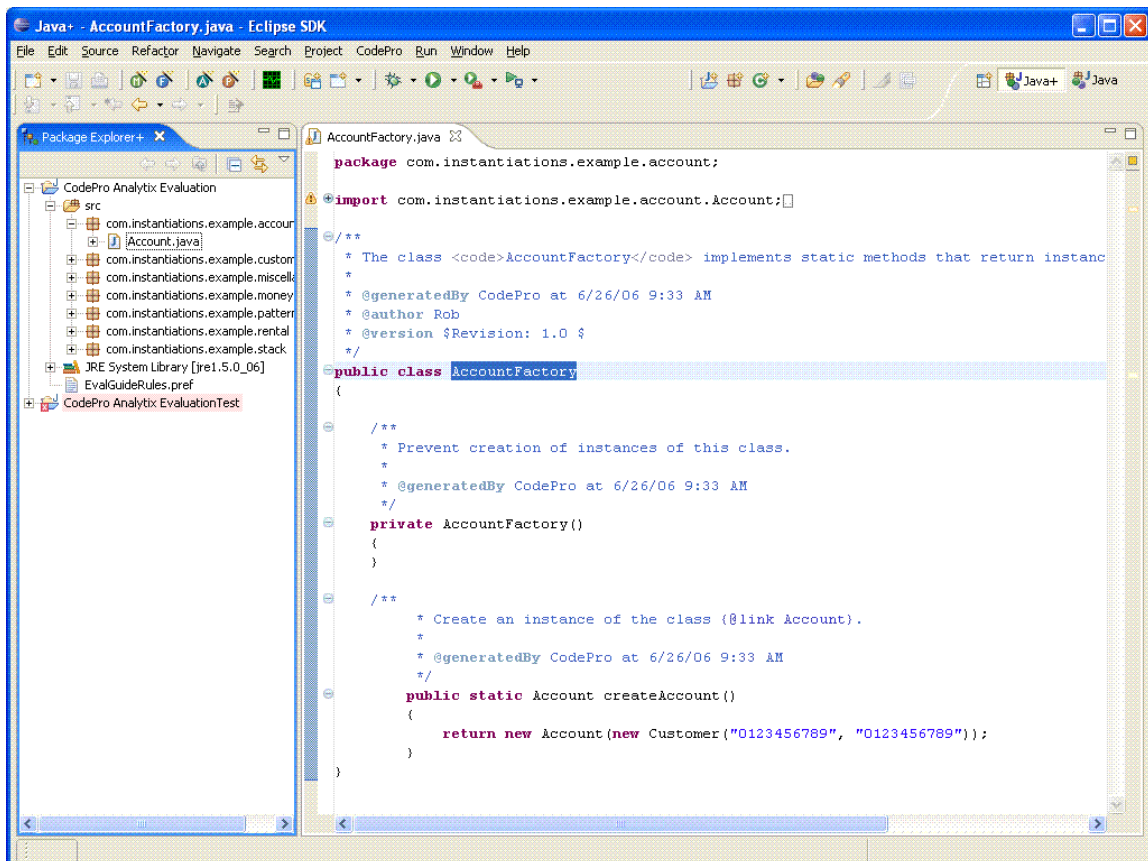
In this section, you will learn how to generate factory classes for testing purposes. Factory classes can be used to improve the quality of the generated JUnit tests.

Features

CodePro AnalytiX offers a flexible Factory Classes option that lets developers specify how to create representative instances of a class for use by the JUnit test generator. This feature will generate a factory class with methods to create the same instances that would normally be produced by the test generator. You can then customize these methods to return better instances.

Steps

1. In the *Package Explorer*, browse to the *Account.java* class (in *src->com.instantiations.example.account*).
2. Right-click on the class and from the *CodePro Tools* sub-menu select *Generate Factory Classes*.
3. When the generation is complete, the tool will open *AccountFactory.java* in the *Editor* view.



4. Verify the content of the factory before closing the *Editor* view.

Benefits

The CodePro AnalytiX JUnit Factory Classes facility provides flexibility and lets developers set options associated with factory classes.

Code Coverage Analysis

Verifying Code Coverage

Goals

In this section, you will learn how to compute the coverage of the test cases generated in a previous step. Code coverage is one indication of how complete the JUnit test cases are. Analyzing the code coverage can help you identify pieces of code that are not being adequately tested.

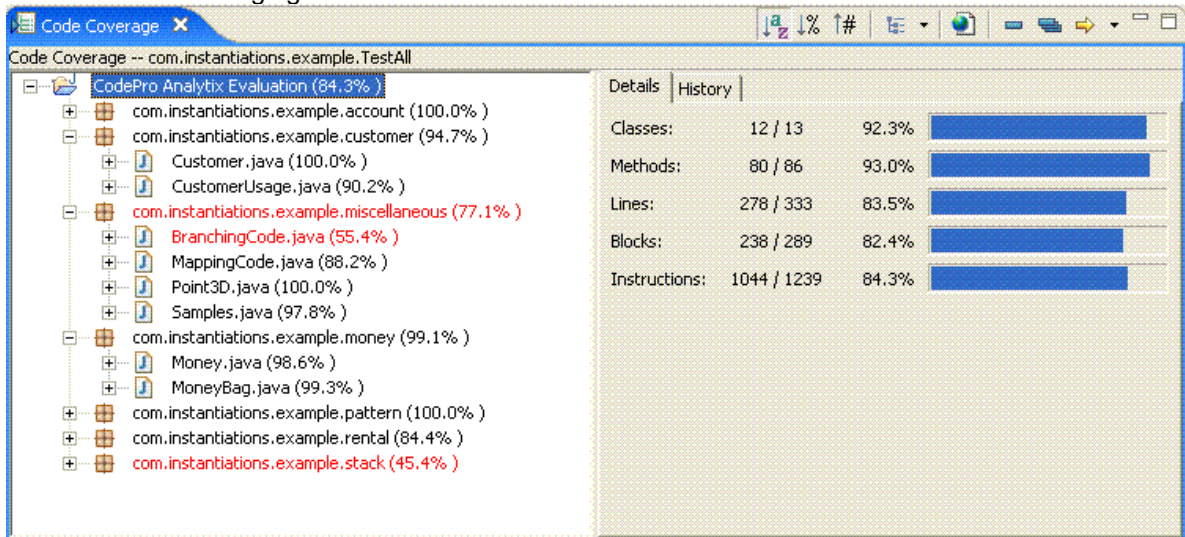
Features

Perhaps the most common reason for measuring code coverage is to evaluate the effectiveness of test code at exercising all possible paths through the code. Using the CodePro AnalytiX Code Coverage facility lets developers easily run code coverage against a collection of test suites or against a manual test script. The tool shows code coverage from the project point of view, the package point of view, or the class point of view. You can even drill down into the individual methods to see exactly which ones are covered or not. For example, you could configure CodePro to flag anything that falls below 80% coverage and mark it in red.

Steps

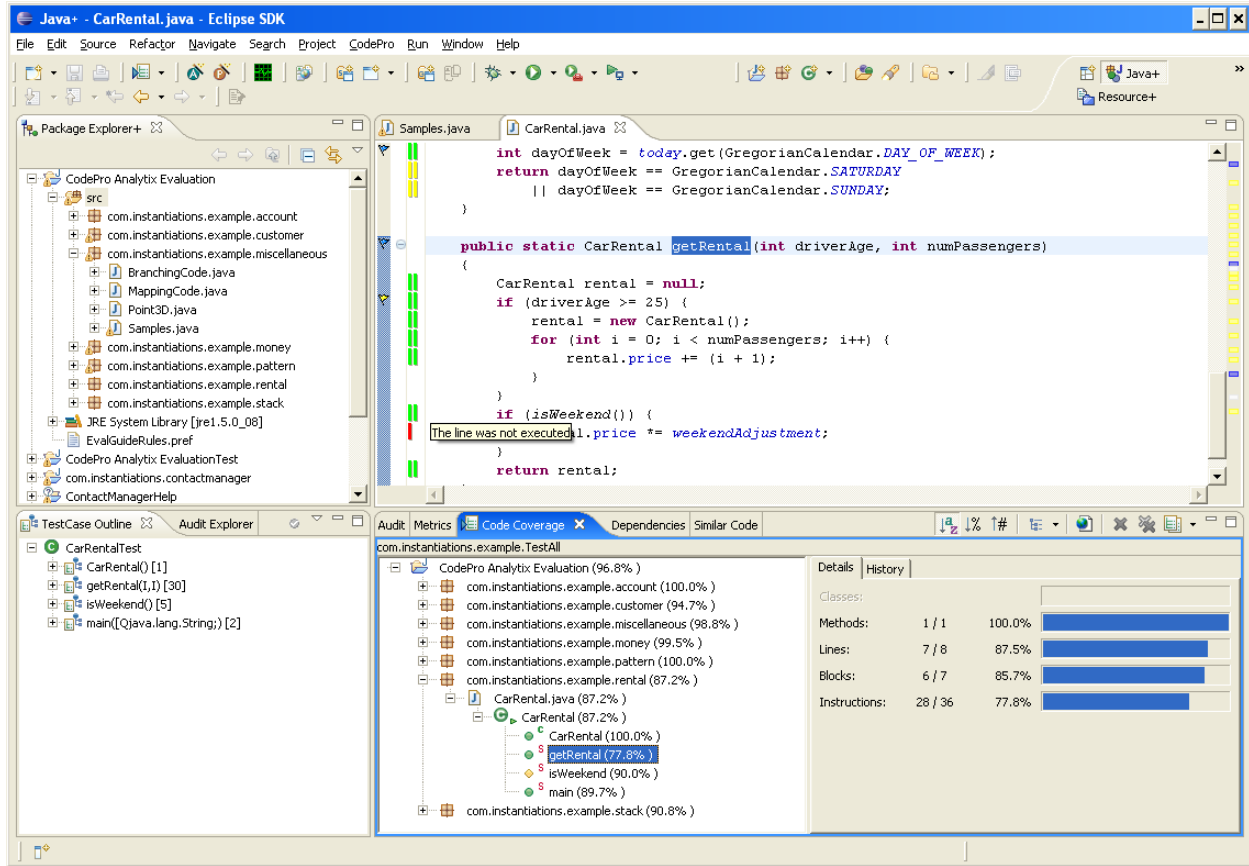
1. In the *Package Explorer*, right-click on the *CodePro AnalytiX Evaluation* project and from the *CodePro Tools* sub-menu select *Instrument Code*.
2. Browse to *TestAll.java* in the *CodePro AnalytiX EvaluationTest* project (under *src -> com.instantiations.example*). Right-click on the class and from the *Run As* sub-menu select *JUnit Test*.

3. After the test execution has completed, the *Code Coverage* view will open to show the code coverage results. Browse the results by expanding the packages to see the code coverage of each class and expanding the classes to see the coverage of individual methods. An example is shown in the following figure.



4. Double-clicking on an individual method will automatically take you to that method, as shown below. The colored bars in the gutter indicate how effective the test execution was on each specific line.

CodePro AnalytiX Evaluation Guide



Benefits

The CodePro AnalytiX Code Coverage facility provides information on which code is being executed. This information can be used in troubleshooting and is valuable in creating higher quality code and in reducing QA testing time.

Reporting

Goals

In this section, you will learn how to generate an HTML report containing the results of a single code coverage analysis. This is useful for reporting to management on the status of a project's testing effort.

Features

Each time a particular class is measured during code coverage, you can generate a coverage report showing the results of the coverage.

Steps

1. Open the *Code Coverage* view if necessary (from the *CodePro* menu select *Code Coverage* on the *Views* sub-menu).
2. Right-click in the *Code Coverage* view and select *Generate HTML Report*.
3. Enter a file name in the *Save As* dialog and then click *Save*.
4. Browse to the saved file location and double-click to open the file in a web browser.
5. Drill-down into the report to see the package, class, and method levels in the report. Note the clicking method takes you to the source code for that method in the report. A sample is shown in the following figure.

CodePro Code Coverage for
This document contains the code coverage results for .

Code Coverage Summary		78.4% coverage		Code Statistics		13 classes, 333 executable lines	
Line Coverage	78.4%	Total Packages	7	Total Files	13	Total Classes	13
Block Coverage	77.2%	Total Executable Lines	333				
Instruction Coverage	76.0%						

Classes

com.instantiations.example.account 100.0% coverage

Name	Executable Lines	Line Coverage	Block Coverage	Instruction Coverage
Account.java	6	100%	100%	100%

com.instantiations.example.customer 81.0% coverage

Name	Executable Lines	Line Coverage	Block Coverage	Instruction Coverage
Customer.java	10	100%	100%	100%
CustomerUsage.java	11	64%	64%	66%

com.instantiations.example.miscellaneous 94.9% coverage

Name	Executable Lines	Line Coverage	Block Coverage	Instruction Coverage
BranchingCode.java	55	98%	98%	98%
MappingCode.java	6	100%	100%	100%
Point3D.java	7	100%	100%	100%
Samples.java	50	90%	90%	93%

com.instantiations.example.money 42.2% coverage

Name	Executable Lines	Line Coverage	Block Coverage	Instruction Coverage
Money.java	29	34%	27%	33%
MoneyBag.java	73	45%	46%	46%

Benefits

CodePro AnalytiX contains a Code Coverage reporting tool that shows details on code coverage and historical information.

Dependency Analyzer

Analyzing Dependencies

Goals

In this section, you will learn how to analyze the dependencies between pieces of code, whether at the project, package, or type level. This is important for understanding the full effect of changes to one portion of the code.

Features

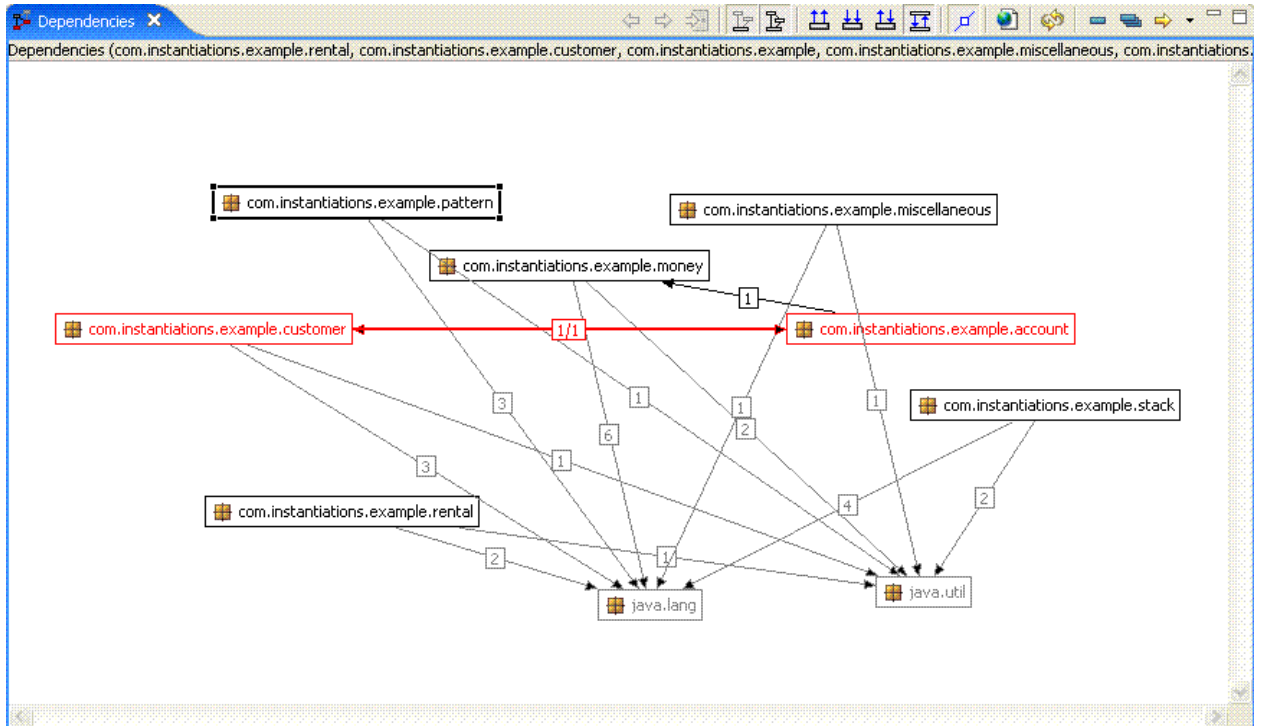
Complex software typically includes code that is invoked by or depends on the behavior of other code. Developers find that it is very time-consuming to find dependencies and determine how the effect of a change in one area will cause changes in other parts of the system. CodePro AnalytiX contains a Dependency Analysis facility that lets developers perform a dependency analysis on one or more projects or packages.

Using CodePro AnalytiX to locate a dependency helps identify where code errors may occur across projects, packages or types and helps determine how a change can affect other code. Dependencies are displayed in a graphical format and color-coding makes it easy to see where circular dependencies exist.

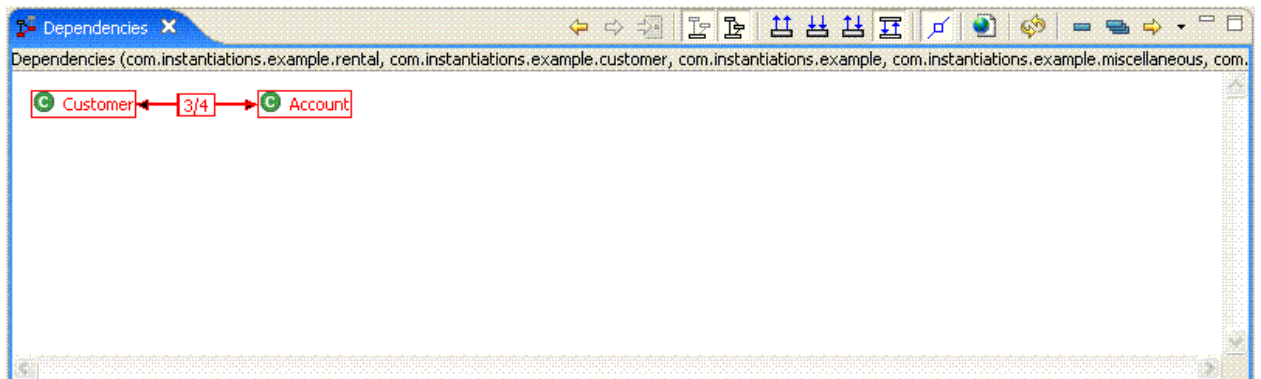
Steps

1. In the *Package Explorer*, expand the *CodePro AnalytiX Evaluation* project, right-click on *src* and from the *CodePro Tools* sub-menu select *Analyze Dependencies*. This will open the *Dependencies* view and display the analysis.

- In the *Dependencies* view, you can arrange the graphical representations of the packages by clicking and dragging them around the view.

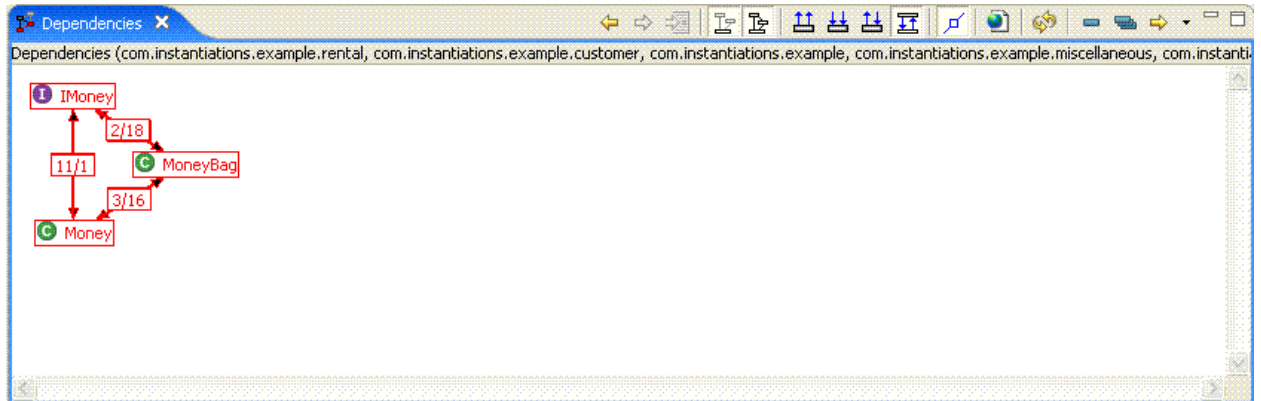


- To drill into the dependency analysis, double-click on the highlighted red 1/1 box between `com.instantiations.example.customer` and `com.instantiations.example.account`. This shows the direct dependencies between the two classes.



- Click the back arrow in the *Dependencies* view to return to the top-level dependency analysis.

5. Double-click on the *com.instantiations.example.money* package to drill into it. This shows an example of a cyclic dependency. Note that the classes and the interface have bi-directional dependencies throughout.



Benefits

Finding code dependencies is important because it helps developers determine how changes will impact other portions of the software and helps prevent propagating errors across multiple projects. CodePro AnalytiX contains a dependency analysis tool that lets developers show dependencies between projects, packages, or types. The tool reduces time spent in locating dependency errors and helps increase the quality of code.

Summary

This Evaluation Guide provides an overview of some of the most commonly used CodePro AnalytiX features. There are a number of additional CodePro AnalytiX features not covered in this guide. For more information, browse the documentation included in the product.